CONFIGURABLE SCHEDULING SYSTEM

CROSS-REFERENCE TO RELATED APPLICATION

This application claims the benefit of U.S. Provisional Applications 60/193,834, 60/193,917, 60/193,832, 60/193,705 and 60/193,833, each of which were filed March 31, 2000, and each of which are incorporated herein by reference

TECHNICAL FIELD

The technical field relates generally to scheduling, and more particularly, to a system and method for a configurable scheduling system that is configurable by a service organization using the system.

COPYRIGHT NOTICE - PERMISSION

BACKGROUND OF THE INVENTION

In many industries which employ a large mobile workforce, such as utilities, taxi companies, and large equipment repair services, the efficiency with which mobile workers can be scheduled and dispatched to customers can have a great impact on customer satisfaction as well as on a service provider's bottom line. From the customer satisfaction standpoint, it is not uncommon for a customer to call a cable television company, or other service provider, to request service only to be told to choose a four-hour service window on

good days, or an "all day" service window on bad days. Even when the customer is "lucky" enough to request service on a "good" day, the worker dispatched by the service provider typically will arrive well after the window has closed, or the customer will have waited, and wasted, most of the day for what should typically only be a half-hour service call. This situation arises from an inability of the service provider to accurately predict when a particular worker will complete a given task and how long it will take for the worker to reach the next service location.

From the financial standpoint, inefficient scheduling and dispatching results in fewer service calls being performed each day, potentially resulting in lower earnings per worker-hour, as well as possible additional expenditures required for hiring and training additional workers. To improve scheduling and dispatching, many service providers have an automated or computerized scheduling system carry out these tasks. Conventional scheduling systems typically perform the scheduling and dispatching tasks through the use of various algorithms that account for many factors in assigning a mobile worker to service a customers work order, such as time availability, skill sets, geographic area, duration of each work order, travel time, and the like. Thus, using scheduling systems such as these allow for a service provider to more efficiently utilize their mobile workers in satisfying customer work orders.

Although scheduling systems have reduced the difficulty a service provider faces in scheduling and dispatching mobile workers, the systems and algorithms employed are generally difficult customize or reconfigure to the needs of a particular service organization. Although some conventional scheduling systems have incorporated some aspects of configurability by the service organization, the extent of configurability is usually limited to customizing the scheduling system by having the service organization select from a set of predefined scheduling rules. However, any additional configuration of the scheduling system by the service organization involves custom reprogramming of the algorithms themselves, which will require the reprogramming be performed the programmers of the scheduling system. Moreover, in the cases where additional

modifications to the scheduling system are required by the service organization, additional reprogramming may be required. As one could imagine, having the algorithms reprogrammed for a custom configuration of the scheduling system can be costly and time consuming. The service organization is also subject to the time constraints of the programmers reprogramming the algorithms. That is, service organizations have no choice but to wait until programmers have time to work on their scheduling systems. For the service organization having limited financial resources, or where reprogramming of the algorithms cannot be completed within the desired timeframe, the service organization is left without any options but to suffer through the use of the existing scheduling system. Therefore, there is a need for a configurable scheduling system having greater flexibility in allowing a service organization using the system to configure the system.

SUMMARY OF THE INVENTION

The present invention relates to a user-configurable scheduling method and apparatus configurable by a service organization for scheduling orders and workers in accordance with a constraint set including a programmable constraint set and a fixed constraint set. The programmable constraint set alters the schedule process from a normal process governed by the fixed constraint set. The constraint set includes programmable rules and constants. The programmable rules may be programmed in accordance with a rule language convention having a first field containing rule identifier to uniquely identify the data structure, and a field containing a rule body including material for altering the schedule process from a standard process to a reconfigured process configured by the service organization.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a system showing the relationship between a service organization, a scheduling system, mobile service representatives, and customers according to and embodiment of the present invention.

Figure 2 is a block diagram showing the scheduling system according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

In the following detailed description of exemplary embodiments of the invention, reference is made to the accompanying drawings, which form a part hereof, and in which are shown, by way of illustration, specific exemplary embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized and structural, logical, electrical, and other changes may be made without departing from the spirit or scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims. The present application incorporates by reference the following references:

Simon Jacobs, Enterprise Scheduling System (ESS) Function Specification, Rev. 4.1 (June 30, 2000); and Simon Jacobs, Configurable WDS (CWDS), Rev. 2.1 (August 1999).

Figure 1 is a block diagram of a system 100 according to one aspect of the present invention. The system 100 includes a service organization 102. The service organization 102 performs services for a roster of customers. When a customer 108 is interested in having a service performed by the service organization 102, the customer 108 calls the service organization 102 to make a reservation for a service to be performed. Using the scheduling system 104, the service organization 102 negotiates with the customer 108 to place a reservation on a schedule. The scheduling system 104 provides to the service organization 102 several appointment windows from which the customer 108 may choose. The customer 108 selects an appointment window in which the service will be performed. During the negotiation, the scheduling system 104 makes preliminary assignment to one or more mobile service representatives 106. A mobile service

representative 106 is one who is capable of performing the service requested by the customer 108. Each time a reservation is placed, the scheduling system 104 accounts for many factors in assigning the reservation to a mobile service representative 106, such as time availability, skill sets, geographic area, duration of each job, travel times between jobs, and equipment requirements. The scheduling system 104 can be implemented over one machine or several machines with different processors. This allows the scheduling system 104 to be scalable depending on the needs of the service organization 102. As will be discussed in more detail below, the scheduling system 104 is also configurable by the service organization, thus allowing the scheduling system 104 to meet different constraints and objectives as defined by the service organization.

Figure 2 illustrates a scheduling system 200 according to an embodiment of the present invention. The scheduling system 200 is similar to the scheduling system 104 shown in Figure 1 and can be substituted for the scheduling system 104. The scheduling system 200 performs a scheduling process. That is, the scheduling process includes appointment negotiation, assignment and optimization processes. It will be appreciated that additional processes may be included, as well as having one or more of the processes described herein omitted, without departing from the scope of the present invention. The processes of appointment negotiation, assignment and optimization have been provided merely for the purposes of an example. It will be appreciated that "process" is used herein to refer to a logical collection of activities.

The scheduling system includes a negotiator 204, an assigner 208, and an optimizer 212 which perform negotiating, assigning, and optimizing functions through the use of a negotiation algorithm 216, an assignment algorithm 220, and an optimization function 224, respectively. Negotiating offers appointment windows to a customer to select for a reservation and has performed enough processing to know that the reservation may be assigned to any of the appointment windows that are offered to the customer. Assigning assigns the reservations to a shift of a mobile service representative provided that the shift and the mobile service representative satisfy certain constraints, such as sufficient skill sets

to perform the work under the reservation. Optimizing changes the way reservations are assigned by the assigning function to meet optimization objectives defined by the service organization. The optimization may move reservations within a shift of a mobile service representative. It may swap reservations between two shifts of two mobile service representatives as well. The optimization is designed to occur independent from the assignment while new reservations are being assigned.

It will be appreciated that scheduling system 200 may be implemented using various computer systems and software techniques that are well-known to those of ordinary skill in the art. A more detailed description of a system in which the scheduling system 200 may be implemented is provided in co-pending U.S. Patent Application _____, entitled ENTERPRISE SCHEDULING SYSTEM FOR SCHEDULING MOBILE SERVICE REPRESENTATIVES, filed on April 2, 2001, to Jacobs *et al.*, which is incorporated herein by reference.

In embodiments of the present invention, operation of the negotiation algorithm 216, the assignment algorithm 220, and the optimization algorithm 224 of the scheduling system 200 is governed by a constraint set, which includes a set of rules and a set of constants. The set of constraints can be defined in a file that is then compiled into a library that are used when the negotiation, assignment, or optimization algorithms are executed. The set of rules include both fixed business rules, which are generally inaccessible by a service organization, and configurable rules that can be programmed by the service organization. Similarly, as will be explained in more detail below, the set of constants are programmable and may be used by the service organization to control the flow of execution of the rules. Their values can be changed at run time by dispatchers and system administrators to adjust the rule set based on different operating conditions. For example, changing the scheduling during a snow store as compared to a sunny day, or scheduling during a major service outage.

Although the underlying structure of the negotiation, assignment, and optimization algorithms 216, 220, and 224 are fixed, and generally inaccessible to the

service organization, the rules, and consequently the constants, are invoked from within the negotiation, assignment, and optimization algorithms 216, 220, and 224 by rule identifiers that are coded into the algorithms. Thus, although the service organization cannot modify the algorithm itself, through the use of the programmable rules and constants, the service organization is provided with a mechanism by which the operation of the negotiation, assignment, and optimization algorithms 216, 220, and 224 of the scheduling system 200 may be customized or configured to the particular needs of the service organization using the scheduling system 200.

Examples of the types of rules and constants shall now be discussed to illustrate their use in customizing or configuring the scheduling system 200. It will be appreciated, however, that the following description, although describing particular embodiments, is provided by way of example, and should not be interpreted in limiting the scope of the present invention.

As mentioned previously, the rules of the constraint set include both fixed business rules and configurable or programmable rules programmed by the service organization. However, the rules, both the fixed and configurable ones, can also be generally categorized by a type as well. For example, the rules of the constraint set can be categorized according to the following types.

Order candidacy rules, which govern which orders can be considered by the appointment, negotiation, assignment, and optimization algorithms;

Mobile user candidacy rules, which govern which mobile users can be considered by the appointment, negotiation, assignment, and optimization algorithms;

Assignment candidacy rules, which determine which order and user pairs constitute valid assignments;

Score rules, which provide a measure of the quality of the assignment or an order to a mobile user, and is used in assignment and optimization as well; and

Intermediate rules refer to those rules that are referenced by other rules. The intermediate rules are used to build up complex high level rules.

The three candidacy rules, namely, the order, mobile user, and assignment candidacy rules return a value of true or fail. The score rules return a numerical value, typically between zero and one. There can be a variety of score rules. For example, there are a number of instances of each type of candidacy rule, where all instances of each type must be true for the order or user to be considered a candidate. It will be appreciated that the various rules previously described can be used or accessed by one or all of the algorithms executing in the scheduling system, including the negotiation, assignment and optimization algorithms. As will be explained in more detail below, certain types of business rule values are available to corresponding types of rules.

As mentioned previously, the grammar for the rule language is designed to provide enough flexibility to express the various business rules that need to be defined for the scheduling system. A programmable rule is written by a service organization according to a language convention, which is subsequently translated into a rule grammar format. The same language convention could be used for the fixed business rules and constraint sets as well, except that the fixed business rules are inaccessible by the service organization. A suggested language convention is as follows.

A programmable rule is identified by a <rule_id> which is a label starting with the symbol "R:". The <rule_id> must be unique within the rule context group. The rule is composed of a rule body or rule clause composed of different rule types or more rule identifiers. The rule clause is bounded by a pair of brackets ({}), and each of the rule identifiers therein are separated by a semicolon (;). The rule is then terminated by a period (.).

In addition to defining the rule language convention, data types are defined as well. The following list provides examples of those data types that can be defined within the language.

- **Rule label.** This label identifies a rule for future reference. Rule labels begin with R:.

- **Variable label.** This type of label is enclosed in [square brackets] and denotes a value that is returned by the system that the rule may access. The value is returned by a function and may be a single field in a database or the result of combining information from a variety of sources (e.g. a synthesis of different database fields). The motivation for providing this label mechanism is to furnish the service organization with the ability to reference values that are logically related to one another (e.g., the wage paid to a given mobile user). A variable label may be composed of one or more elements, each separated by a period (.). Each of the elements denotes a sub-field of those to its left; for example, [order.date] denotes the date belonging to the order object, and [order.assignedUser.state] denotes the state of the mobile user assigned to the order.

Variables may, as will be discussed in more detail below, represent literals, numbers, dates and strings. If the variable label is preceded by a dollar sign ($) the label is taken to be a database field name and not an accessor function. This is done in order to allow for the possibility that a service organization creates a customized database field or configurable fields that can be used by the rules and constants. For example, some configurable fields include customer contact number, completion codes (showing work done and parts or materials used to complete and order), customer account number, previous work history, and the like.

- **Dates.** Dates are bounded by a pair of percentage signs (%%) and are specified in the following format:

yyyy/mm/dd (numeric year, month, and day)

- **Times.** Times are bounded by a pair of ampersand signs (&&) and are specified in the following format:

hh:mm:ss (hours, minutes, and seconds)

- **Datetimes.** Datetimes are bounded by a pair of "at" signs (@@) and are specified in the following format:

yyyy/mm/dd-hh:mm:ss

- **Numbers.** These may be signed or unsigned, integers or real numbers base ten. Real numbers are specified in floating point format only.

- **Strings.** These are delimited by a pair of double quotes ("") and may contain alphanumeric or special characters.

- **Literals** may be any of the date, number or string types. A literal label represents a label that has been equated to a literal. Literal labels are delimited by a pair of colons (::). Some special literals are defined for the rule language:

- **Boolean values** are treated as literals *e.g.*, :true: and :false:. An undefined or non-existent value is :nil:

- **Sets** are supported in so far as there are operators to perform membership comparisons on sets. There is no defined mechanism for defining a set literal (via a literal label).

Based on the rule semantics and data types previously defined, an example programmable rule is provided:

R:o11 {[order.state] eq :pending:}

is a Boolean rule where "R:o11" is the rule label and "[order.state] eq :pending:" is the rule clause. Within the rule clause, "order.state" is a variable label which provides the current state of the order, and ":pending:" is a literal label. Essentially, the rule R:o11 will return a "true" value for all the orders that are currently pending. As will be shown in various examples below, simple rules such as R:o11 can be combined with other rules to configure the operation of the negotiation, assignment, and optimization algorithms.

The rules written according to the rule language convention previously described are subsequently translated into a rule grammar. The following list provides an example rule grammar as expressed using a BNF notation. That is, the expression to the

left of the "::=" symbol are expanded to the expression provided to the right. Additionally, optional selection of a terminal or non-terminal symbol is indicated with an OR symbol "|", where terminal symbols are in **bold** and non-terminal symbols are enclosed in angle brackets "<>".

Rule definition:

```
<rule>          ::=  <rule_id> <rule_body>
<rule_id>       ::=  R: <id_label> <space>
<rule_body>     ::=  <allof_rule> | <anyof_rule> | <oneof_rule> |
                          <cond_rule> | <expr_rule> | <exec_rule>
<allof_rule>    ::=  all-of  <blnk> { <blnk> <allof_body> <blnk> }
<allof_body>    ::=  <rule_id> | <rule_id> ; <allof_body>
<anyof_rule>    ::=  any-of  <blnk> { <blnk> <anyof_body> <blnk> }
<anyof_body>    ::=  <rule_id> | <rule_id> ; <anyof_body>
<oneof_rule>    ::=  one-of  <blnk> { <blnk> <oneof_body> <blnk> }
<oneof_body>    ::=  <rule_id> | <rule_id> ; <oneof_body>
<exec_rule>     ::=  exec-fn <blnk> { <id_label> }
<expr_rule>     ::=  { <blnk> <exp_body> <blnk> }
<exp_body>      ::=  <rule_id> | <rule_id> ; <exp_body>
<cond_rule>     ::=  if <blnk> { <blnk> <cond_clause> <blnk> } <blnk>
                     then
                     <blnk> { <blnk> <expr_clause> <blnk> } <blnk>
                     else
                     <blnk> { <blnk> <expr_clause> <blnk> }
<cond_clause>   ::=  <clause_body> | <rule_id>
<expr _clause>  ::=  <expr> | <rule_id>
```

Rule clause definition:

```
<rule_clause>   ::=  <rule_id> { <rcl_body> }.
<rcl_body>      ::=  <blnk> <rcl_item> <blnk>
<rcl_item>      ::=  <clause_body> | <rule_body> | <rule_id>
<clause_body>   ::=  <value><logop><value>|<setval><setop><setval>
```

Expression definition:

```
<expr>          ::=  <num_exp> | <date_exp>
<num_exp>       ::=  <numval>|(<blnk><num_exp><numop><num_exp><blnk>)
<date_exp>      ::=  <dateval>|(<blnk><dateval><dateop>
                     <numval><blnk>)
<numval>        ::=  <number> | <literal> | <variable>
<dateval>       ::=  <date> | <literal> | <variable>
<value>         ::=  <blnk>  <val_type>  <blnk>
<setval>        ::=  < <set_body> >
<set_body>      ::=  <set_element> | <set_element> <blnk> <set_body>
<set_element>   ::=  <number> | <date> | <literal>
<val_type>      ::=  <number>  |  <string>  |  <date>  |  <literal>  |
                     <variable> | <rule_id>
<literal>       ::=  : <id_label> :
<variable>      ::=  [ <var_label> ]  |  [ <var_label_1>]
<var_label_1>   ::=  $<var_label>
<var_label>     ::=  <id_label> | <var_label> . <var_label>
<id_label>      ::=  <alpha> <alphanum>
<alphanum>      ::=  <alpha> | <numeric>
<string>        ::=  " <string_body> "
<string_body>   ::=  <ans> | <string_body> <ans>
<number>        ::=  <sign> <real> <space> | <sign> <integer> <space>
<date>          ::=  % <date_defn> %
```

```
<date_defn>     ::=  <yyyy> / <mon_mon> / <dd>
<time>          ::=  & <time_defn> &
<time_defn>     ::=  <hh> / <min_min> / <ss>
<datetime>      ::=  @ <date_defn> - <time_defn> @
<real>          ::=  <integer> . <floatpart>
<floatpart>     ::=  <null> | <integer>
<integer>       ::=  <numeric> | <integer> <numeric>
<ans>           ::=  <alpha> | <number> | <special>
<sign>          ::=  <null> | + | -
<dd>            ::=  <integer>     Range: 01 -31
<mon_mon>       ::=  <integer>     Range: 01 -12
<yyyy>          ::=  <integer>     Range: 0 -9999
<hh>            ::=  <integer>     Range: 0 - 23
<min_min>       ::=  <integer>     Range: 0 - 59
<ss>            ::=  <integer>     Range: 0 - 59
<logop>         ::=  eq | neq | lt | leq | gt | geq |
<setop>         ::=  seq | mem | nmem | in | notin
<numop>         ::=  + | - | * | / | **
<dateop>        ::=  + | -
<alpha>         ::=  a | b | c | d | e | f | g | h | i | j | k | l |
                     m | n | o | p | q | r | s | t | u | v | w | x |
                     y | z |
                     A | B | C | D | E | F | G | H | I | J | K | L |
                     M | N | O | P | Q | R | S | T | U | V | W |
                     X | Y | Z
<numeric>       ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<special>       ::=  ! | @ | # | $ | % | ^ | & | * | / | + | _ |
<space>
<blnk>          ::=  <spaces> | <null>
<spaces>        ::=  <space> | <spaces> <space>
<space>         ::=  UNICODE space character
<null>          ::=  No token required
```

The operators used in the above grammar are defined as follows.

all-of

This operator applies to all clauses identified in the rule body. A logical **AND** operation is performed on the clauses, thus:

R:xyz **all-of** { R:o1; R:o2; R:o3 }.

is equivalent to:

R:o1 **AND** R:o2 **AND** R:o3

The value returned by this operator is a Boolean (true or false).

any-of

This operator applies to all clauses identified in the rule body. A logical **OR** operation is performed on the clauses, thus:

R:xyz **any-of** { R:o1; R:o2; R:o3 }.

is equivalent to:

R:o1 **OR** R:o2 **OR** R:o3

The value returned by this operator is a Boolean (true or false).

one-of

This operator applies to all clauses identified in the rule body. A logical exclusive OR (**XOR**) operation is performed on the clauses, thus:

R:xyz **one-of** { R:o1; R:o2; R:o3 }.

is equivalent to:

R:o1 **XOR** R:o2 **XOR** R:o3

The value returned by this operator is a Boolean (true or false).

Expression rule

An expression rule may consist of an expression that can either be a numeric, date or string expression or may consist of a number of rule identifiers. The result of the execution of the rule is the value of the expression or, in the case of a number of rules, the value returned by the last rule executed. An "if" statement without an "else" clause is only considered to execute if the condition is true. Thus, the rule:

R:xyz { R:o1; R:o2; R:o3 }.

executes each of the rules and returns the value of the last rule that executed. Thus, if no conditional clauses are evaluated the result of executing rule R:xyz will be the result returned by rule R:o3. The expression rule provides a flexible mechanism for performing a number of conditional clauses where the rule is expected to yield a value corresponding to the conditional clause that succeeded.

exec-fn

This operator applies to a single named function in the rule body. The result returned by the rule is that returned by the named function. This operator is provided so that complex expressions may be invoked as a single rule clause without requiring the service organization to program the expression.

Conditional clause

Conditional expressions only have a consequent and "else" clause. The former is evaluated if the logical expression of the conditional evaluates to true otherwise the "else" expression is evaluated. The syntax used is the following:

If clause **then** expression **else** expression.

The clause may be a simple Boolean clause or may be a rule identifier. The expression may be numeric (returning an integer or real value), a date expression (returning a date) or a rule identifier.

Logical Operators (A op B)

Logical operators must be provided with operands of the same type. In the event that one or both operands are undefined (:nil:) the result of the operation will be :false:

In the following expressions, if A and B evaluate to numeric or date values the comparison is done based on the arithmetic relationship between the operands. If A and B evaluate to strings the comparison is done according to the UNICODE collating sequence.

eq	Returns true if $A = B$
neq	Returns true if $A \neq B$
leq	Returns true if $A \leq B$

geq	Returns true if $A \geq B$
lt	Returns true if $A < B$
gt	Returns true if $A > B$

Set Operators (A op B)

In the following expressions, A and B must represent sets of similar objects.

mem	Returns true if $A \cap B \neq \varnothing$
nmem	Returns true if $A \cap B = \varnothing$
seq	Returns true if $A = B$
in	Returns true if $A \in B$
notin	Returns true if $A \notin B$

The following tables describe valid label sets that can be used in the grammar for the rule language, given the type of rule being created. The tables presented below list the variable labels that the service organization is allowed to reference and the values that the variables referenced may take.

Order Candidacy and Assignment Candidacy. The following table provides some examples of the types of business rule values that may be used in expressions for order candidacy and assignment candidacy:

<var_label>	Type	Value
order.state	Literal	en-route | pending | on-site | dispatched | complete | cancelled | suspended
order.isAssigned	Literal	true | false
order.scheduleDate	Date	date | nil
order.isAutoDispatchable	Literal	true | false
order.priorityType	Literal	normal | emergency
order.context	Literal	current | future | undated
order.assignedUser	Label	Mobile User identifier

order.area	Variable	The area for the order
order.amountCollected	Number	Amount collected

Mobile User Candidacy and Assignment Candidacy. The following table enumerates the business rule values that may be used in expressions for mobile user candidacy and assignment candidacy:

<var_label>	Type	Value
user.isAutoDispatchable	Literal	true | false
user.state	Literal	available | unavailable | en-route | on-site | emergency
user.isInEmergencycontrol	Literal	true | false
user.area	Variable	The mobile user's assigned area
user.wage	Number	mobile user wage

Assignment Candidacy. The following table enumerates the values that may be used in expressions for assignment candidacy:

<var_label>	Type	Value
user_order.travelTime	number	mobile user travel time

Area Data. The following table enumerates some of the values that may be used in expressions involving areas:

<var_label>	Type	Value
area.travelMin	number	Minimum area travel time
area.travelMax	number	Maximum area travel time

System Data. The following table enumerates some of the system values that may be used in expressions for mobile user or order candidacy:

<var_label>	Type	Value
SystemDate	date	Date
SystemMinUserwage	number	Minimum mobile user wage
SystemMaxUserwage	number	Maximum mobile user wage
SystemAmountCollectedMin	number	Minimum amount collected
SystemAmountCollectedMax	number	Maximum amount collected
SystemSchedulingActivityType	literal	scheduled | redistribution | in-day | floater|suggest | forecast | auto-distribute

Constants, as briefly discussed above, allow a service organization to control the flow of execution within the rules. Constants are a mechanism for replacing global constants that would otherwise require explicit reference in the programmable business rules. This feature is illustrated by the following example.

Let there be a Boolean constant **X** that the service organization sets to establish some configuration. Additionally, the service organization has included a programmable rule that should be invoked if the value of **X** is true. Through the use of constants, the rule could be formulated in the form:

Rule R1: If (**X** = true) and (**Y** or **Z**) then expression.

This rule could be recast in the following form, using a constant:

Constant C1: true.

Rule R2: If (C1) and (**Y** or **Z**) then expression.

Thus, this mechanism allows the service organization to change the behavior of rules by manipulating secondary level rules (those invoked explicitly by other rules) rather than referencing a global value explicitly.

The following are some examples illustrating the use of the programmable rules and constants that may be created by the service organization using the rule language

convention previously described. It will be appreciated that the following examples have been provided to illustrate various aspects of the present invention, and are not intended to limit its scope.

Constants:

Emergency Order Dispatching:

This constant allows for emergency order dispatching. It is checked in the order candidacy rule.

R:EmergencyOrderDispatching {{**:true:**}}

If we wish to turn off emergency order dispatching the constant would read thus:

R:EmergencyOrderDispatching {{**:false:**}}

Rules:

Order Candidacy Rules:

This example uses the constant rule **R:**EmergencyOrderDispatching previously defined. The order candidacy rule (R:OrderCandidacy) describes the conditions under which an order can be automatically assigned by the scheduling system to a mobile user. For example, for an order to be assignable, the order state must be pending (R:o11) or dispatched (R:o12), the order must be enabled for automatic assignment (R:o6), as well as a number of other conditions. The order candidacy rule also makes use of a constant. For example, there is a constant that indicates whether an emergency order can be automatically dispatched (R:EmergencyOrderDispatching). If the value of this constant is false, that is, emergency orders cannot be automatically dispatched, and the order is an emergency order, then it cannot be automatically assigned. However, if the value of this constant is true, then the order can be dispatched even it is an emergency order.

R:OrderCandidacy **all-of** { **R:**o1; **R:**o3; **R:**o4; **R:**o6; **R:**o7; **R:**o9 }.

R:o1 {**any-of** { **R:** o11; **R:** o12 }}.

R:o11 {[order.state] **eq** :pending:}.

R: o12 {[order.state] **eq** :dispatched:}.

R:o3 {[order.isAssigned] **eq** :false:}.

R:o4 {[order.scheduleDate] **le** [systemDate]}.

R:o6 {[order.isAutoDispatchable] **eq** :true:}.

R:o7 {**any-of** { **R:**o71; **R:**o72}}.

R:o71 {[order.priorityType] **neq** :emergency:}.

R:o72 {**all-of** {**R:**o721; **R:**EmergencyOrderDispatching}}.

R:o721 {[order.priorityType] **eq** :emergency:}.

R:o9 {**R:**Usercandidacy}.

Mobile User Candidacy Rules:

The mobile user candidacy rule (R:UserCandidacy) describes the conditions under which a mobile user can be automatically assigned orders. For example, to be assigned orders, a mobile user must be enabled for automatic assignment (R:f100) and must not be in an emergency state (R:f101), as well as a number of other conditions.

R:UserCandidacy **all-of** { **R:**f100; **R:**f101; **R:**f102; **R:**f105 }.

R:f100 {[order.assignedUser.isAutoDispatchable] **eq** :true:}.

R:f101 {[order.assignedUser.state] **neq** :emergency:}.

R:f102 {[order.assignedUser.isInEmergencyControl] **neq** :true:}.

R:f105 {**any-of** {**R:**f1051; **R:**f1052}}.

R:f1051 {[order.assignedUser.mode] **neq** :local:}.

R:f1052 {**all-of** {**R:**f10521; **R:**UserlocalModeDispatch }}.

R:f10521 {[order.assignedUser.mode] **eq** :local:}.

Amount Collected Rule:

The amount collected rule provides a score, between 0 and 1, that indicates the relative value of (that is, the revenue generated by) working on an order. Higher values represent more revenue collected. Note that rule **R:**ac10 is used in many places in the following example. Thus, the **R:**ac10 rule provides a good example of the use of an intermediate rule.

R:AmountCollectedScore { **R:**ac1; **R:**ac2; **R:**ac3 }.

R:ac1 {**if** {**R:**ac10 **leq** 0} **then** { 0 }}.

R:ac2 {**if** { **R:**ac20 }} **then** { **R:**10 }}.

R:ac20 **all-of** { **R:**ac21; **R:**ac22 }.

R:ac21 { **R:**ac10 **gt** 0 }.

R:ac22 { **R:**ac10 **lt** 1 }.

R:ac3 {**if** {**R:**ac10 **geq** 1} **then** { 1 }}.

R:ac10 { **if** {[systemAmountCollectedMax] **eq** [systemAmount CollectedMin]} **then**

{ 1 } **else** {**R:**ac101}}.

R:ac101 {(1 – (([order.amountCollected] – [systemAmount CollectedMin]) / ([systemAmountCollectedMax] – [systemAmount CollectedMin])))}.

Mobile User Wage Rule:

The mobile user wage rule provides a score, between 0 and 1, that indicates the whether the mobile user is expensive to assign to orders. Lower values represent more expensive mobile users.

R:WageScore { **R:**w1; **R:**w2; **R:**w3 }.

R:w1 {**if** {**R:**w10 **leq** 0} **then** { 0 }}.

R:w2 {**if** { **R:**w20 }} **then** { **R:**10 }}.

R:w20 **all-of** { **R:**w21; **R:**w22 }.

R:w21 { **R:**w10 **gt** 0 }.

R:w22 { **R:**w10 **lt** 1 }.

R:w3 {**if** {**R:**w10 **geq** 1} **then** { 1 }}.

R:w10 {**if** {[systemMaxUserWage] **eq** [systemMinUserWage]} **then** { 1 } **else** {**R:**w101}}.

R:w101 {(1 – (([user.wage] – [systemMinUserWage]) / ([systemMaxUserWage] – [systemMinUserWage])))}.

It will be appreciated that certain details have been set forth herein to provide a sufficient understanding of the invention. However, it will be clear to one skilled in the art that the invention may be practiced without these particular details. In other instances, well-known circuits, control signals, timing protocols, and software operations have not been shown in detail in order to avoid unnecessarily obscuring the invention. Additionally, computer and/or software implementation of embodiments of the present invention are well-known to those of ordinary skill in the art. It will be further appreciated that, although specific embodiments of the invention have been described herein for purposes of illustration, various modifications may be made without deviating from the spirit and scope of the invention. Accordingly, the invention is not limited except as by the appended claims.